IPC@CHIP+CoDeSys
или маленькие ядра для большой артиллерии 3

Пастушенков Д.В

Итак, мы делаем большой шаг и переходим в мир CoDeSys, как и было обещано во второй части нашей статьи [1]. Именно в мир CoDeSys, а не IPC@CHIP, потому что работа в CoDeSys для любого контроллера практически одинакова - именно в этом заключается цель CoDeSys и стандарта МЭК 61131-3. Не даром лозунгом организации CoDeSys Automation Alliance (См. рис.1), объединяющей изготовителей оборудования для промышленной автоматизации, является фраза “One world.One tool” (Один мир. Один инструмент).

Но все-таки у BECK IPC@CHIP есть одна очень важная особенность, о которой мы и поговорим.

Рис. 1 CoDeSys Automation Alliance

Идеология CoDeSys

CoDeSys состоит из двух частей: системы программирования и системы исполнения. Система программирования – это приложение Windows, которое позволяет создавать и отлаживать приложения на языках МЭК. Система исполнения – это программа, которая выполняется в целевом устройстве и поддерживает базовые функции системы программирования, отладку, визуализацию и самое главное процесс выполнения МЭК-приложения. В случае с BECK IPC@CHIP связь между системой программирования и системой исполнения осуществляется по RS-232 или Ethernet.

Проект, созданный с помощью CoDeSys, в принципе, может выполняться на любом контроллере, поддерживающем CoDeSys. Но как именно система программирования узнает на каком контроллере будет выполняться приложение? Ведь разные ПЛК имеют разные процессоры, разный набор входов/выходов и т.д. Для этого используется так называемый *Target support package* (TSP) – набор конфигурационных файлов, которые дают системе программирования полную информацию о контроллере. В системе программирования CoDeSys может быть установлено сразу несколько TSP для контроллеров различных производителей. Таким образом, один раз написав приложение на языках МЭК, мы можем загружать его в разные контроллеры, используя при этом разные TSP. Далее в статье приводиться пример приложения управления движением на перекрестке с помощью контроллера BECK IPC@CHIP. Но ни что не мешает запустить этот пример на любом другом контроллере с CoDeSys.

Таким образом, что такое CoDeSys для конечного пользователя? Пользователь покупает контроллер, ему в придачу дают диск с CoDeSys и соответствующим TSP, пользователь устанавливает систему программирования и TSP, пишет программы для ПЛК и тем самым решает свои прикладные задачи. Все достаточно просто. Система программирования бесплатна и доступна для обновления свободно.

Что такое CoDeSys для производителя контроллера? Тут дело обстоит несколько сложнее. Производитель контроллера должен сначала купить систему исполнения в исходных текстах, адаптировать ее под свое железо, подготовить TSP и только после этого отдать контроллер, уже с поддержкой CoDeSys, конечному пользователю. Процесс адаптации не дешев и требует привлечения высококвалифицированных специалистов. В итоге реализовать поддержку CoDeSys в своих контроллерах могут только достаточно крупные компании, имеющие значительный годовой объем производства. Для изделий выпускаемых на заказ и в малых объемах адаптация нерентабельна.

В данной ситуации решение фирмы BECK выглядит революционным. Вместе с IPC@CHIP фирма BECK IPC GmbH бесплатно поставляет специальную программу *IEC Platform Builder*, которая позволяет создать систему исполнения CoDeSys и набор конфигурационных TSP файлов под конкретное оборудование, построенное на базе BECK IPC@CHIP (См. рис.2).

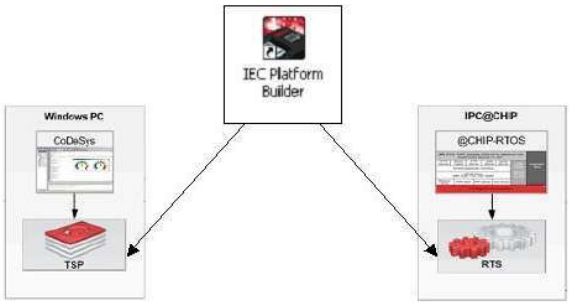

Рис.2. Архитектура CoDeSys для BECK IPC@CHIP

Часть системы исполнения отвечающая за работу с внешними аппаратными средствами включена в исходных текстах на языке С. В результате, мы получаем возможность дооснастить IPC@CHIP любыми устройствами (внешние АЦП, счетчики, удаленные модули ввода и т.п.) и включить их поддержку в CoDeSys. Наш пользователь получит контроллер с готовой системой программирования. Ни каких дополнительных драйверов или специальных библиотек ему не нужно. Все поддержка железа уже реализована.

Такое решение фирмы BECK открывает возможность даже небольшим фирмам создать собственное устройство с полноценной поддержкой CoDeSys.

Подробно изучать IEC Platform Builder мы не будем – это выходит за рамки данной статьи. Давайте рассмотрим по шагам работу CoDeSys с уже готовыми устройствами на базе IPC@CHIP. В качестве примера возьмем эволюционную плату DK50 производства фирмы BECK IPC GmbH.

Мы постараемся дать первое представление о базовых возможностях CoDeSys, которые будут полезны при работе с BECK IPC@CHIP. Желательно, чтобы читатель был знаком со второй частью данной статьи [1], в которой описаны первые шаги по использованию файловой системы BECK IPC@CHIP и настройке основных параметров.

Шаг первый. Установка файлов целевой платформы на ПК

Первое что нужно сделать это установить систему программирования CoDeSys. Ее можно найти на диске, который поставляется вместе с чипом или на сайте http://www.3s-software.ru. Убедитесь, что устанавливаемая версия не ниже чем 2.3.4.1.

Теперь приступим к установке конфигурационных файлов целевой платформы. Для этого нам понадобиться TSP для BECK IPC@CHIP.

Установка каждого такого TSP одинакова. Мы будем использовать пример TSP DK51_WV, поставляемый в комплекте с платой DK50. Данный TSP включает поддержку Ethernet и Web-визуализации. Его можно найти на диске из комплекта DK51 либо скачать с сайта http://www.beck-ipc.com/codesys.

Установка TSP очень проста – достаточно запустить командный файл *install.bat* из папки tsp.

Шаг второй. Установка системы исполнения для BECK IPC@CHIP

Система исполнения CoDeSys – это программа, которая выполняется в контроллере (целевом устройстве). Она отвечает за связь системой программирования, управляет МЭК - приложением, созданным с помощью CoDeSys и всей периферией устройства на базе IPC@CHIP.

Пример системы исполнения для DK50 называется достаточно легкомысленно *myrts.exe* и находится в том же пакете разработчика, что и TSP.

Скопируйте этот файл на диск a: или b: IPC@CHIP. Использование внешнего flash –диска (b:) позволит работать с web–визуализацией CoDeSys.

Затем нужно прописать запуск *myrts.exe* в *autoexec.bat*. Например, если вы используете внешний диск, то *autoexec.bat* должен выглядеть следующим образом:

```
Extide      ; подключение драйвера диска
b:          ; переход на диск b:
myrts       ; запуск системы исполнения
```

Таким образом, система исполнения будет запущена после включения питания IPC@CHIP. Так как система исполнения выполняется под управлением многозадачной операционной системы RTOS, то параллельно с системой исполнения могут выполняться и другие программы.

Шаг третий. Создание приложение для BECK IPC@CHIP

Теперь перейдем к самому важному шагу – созданию приложения для BECK IPC@CHIP. Для этого рассмотрим простой пример управления движением на перекрестке. Возможно, вам будет удобнее иметь исходные тексты данного примера. Их можно скачать с сайта, посвященного CoDeSys http://www.3s-software.ru.

Постановка задачи

Есть перекресток двух дорог. Движение автомобилей на перекрестке управляется двумя светофорами, каждый из которых разрешает или запрещает движение автомобилей по одной из

дорог. Светофоры работают в противофазе – то есть когда светофор А разрешает движение светофор В запрещает.

Обычно интенсивность движения по каждой из дорог различна в разное время суток. Например, если интенсивность движения по дороге А больше, чем по дороге В, то совершенно логично сделать так, чтобы зеленый сигнал светофора А горел дольше, чем у светофора В. Это позволит повысить пропускную способность перекрестка. Правда водители автомобилей на дороге В будут не очень счастливы – придется постоять на перекрестке немного дольше.

Было бы хорошо, если бы период разрешающего светофора менялся динамически в зависимости от текущей ситуации на дороге.

Можно предложить несколько способов выбора периода действия разрешающего сигнала каждого светофора. Во-первых, можно составить расписание, в котором указать режим работы светофоров в разные времена суток. Во-вторых, интенсивность движения можно измерять и периодически на основе полученных данных, задавать какой из двух светофоров будет иметь более длительный разрешающий сигнал.

Но не будем слишком усложнять задачу (она все таки учебная) и воспользуемся самым простым способом - будем задавать периоды разрешающего сигнала в ручную.

В итоге, после того как мы сделаем этот важный шаг, мы получим возможность управлять светофором (включать/выключать, задавать периоды действия разрешающего сигнала) удаленно, например через Интернет.

Создание проекта

Открываем CoDeSys, в меню *File* выполняем команду *New*. Теперь нужно выбрать целевую платформу (устройство), в которой будет выполняться наше приложение. Выбираем DK51_WV – это плата DK50 с поддержкой web-визуализации (рис. 3).

Рис.3. Выбор целевой платформы

После этого появляется диалоговое окно, в котором нам прелагается создать новый программный компонент (POU). По умолчанию это PLC_PRG – программа, вызываемая в каждом рабочем цикле контроллера. Выбираем язык SFC и соглашаемся с прочими опциями диалога *New POU* (рис. 4).

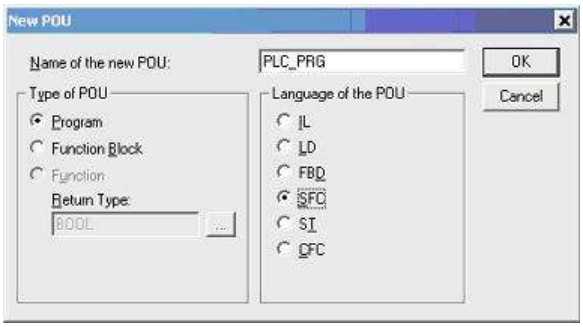

Рис.4. Создание программы PLC_PRG

Функциональный блок TRAFFICSIGNAL

Но на время забудем про PLC_PRG – будем строить приложение снизу вверх. Создадим функциональный блок TRAFFICSIGNAL, который будет отвечать за работу одного светофора. Функциональный блок будет иметь следующий интерфейс (рис.5)

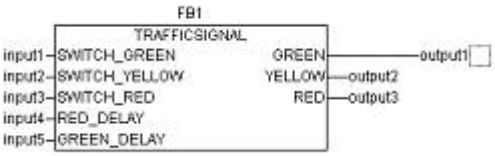

Рис. 5. Функциональный блок TRAFFICSIGNAL

Вход SWITCH_GREEN включает зеленый сигнал светофора, SWITCH_YELLOW – желтый, SWITCH_RED – красный. После включения зеленого сигнала он будет гореть в течение времени GREEN_DELAY, а красный сигнал горит в течение времени RED_DELAY. Время работы желтого сигнала не подается на вход, так как оно фиксировано и составляет 2с.

С выходами все понятно – GREEN, YELLOW, RED – включают сигналы соответствующих цветов.

Теперь опишем логику работы данного функционального блока на языке FBD. Чтобы управлять светофором нам понадобится таймер TP, который входит в библиотеку *standard.lib*. Эта библиотека присоединяется к проекту автоматически при его создании.

Логика работа таймера показана на рис. 6.

То есть вход IN включает выход Q на время PT. Время с момента включения выхода индицируется на выходе ET.

© Компоненты и технологии N7 2005г

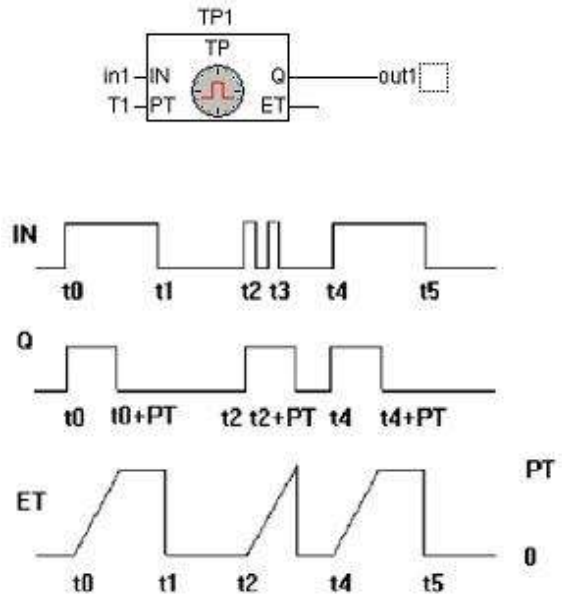

Рис. 6. Логика работы таймера TP

Используя такой таймер, работу светофора можно описать с помощью трех FBD схем (рис. 7).

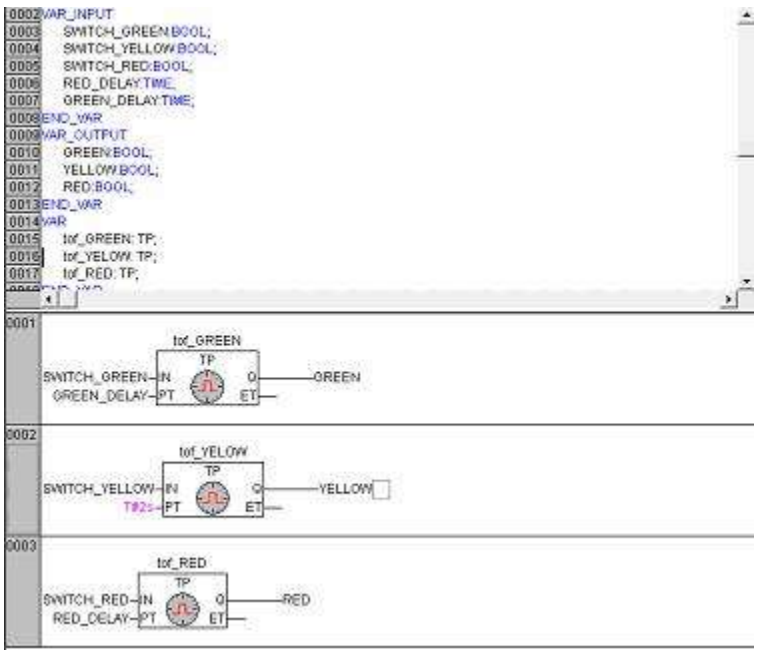

Рис. 7. Логика работы функционального блока TRAFFICSIGNAL

Функциональный блок состоит из двух разделов: из раздела объявлений и из схем, которые описывают логику его работы. В разделе объявления описываются входы/выходы функционального блока, а так же внутренние переменные. В данном случае внутренними переменными являются экземпляры функциональных блоков TP – по одному на каждый сигнал.

С работой функционального блока, наверное, все понятно.

PLC_PRG

Созданный нами функциональный блок TRAFFICSIGNAL позволяет лишь включать и выключать сигналы светофора. Теперь наша задача задать последовательность включения сигналов каждого светофора. Заметим, что сигналы одного светофора зависят от сигналов другого.

Для решения этой задачи будем использовать упрощенный язык SFC – он как раз подходит для задач, в которых действия выполняются последовательно.

```
0001 PROGRAM PLC_PRG
0002 VAR
0003     TS1:TRAFFICSIGNAL;
0004     TS2:TRAFFICSIGNAL;
0005
```

Рис.8. Объявления TS1 и TS2

Возвращаемся к программе PLC_PRG, которую мы создали в самом начале.

В разделе объявлений нужно объявить 2 экземпляра функционального блока TRAFFICSIGNAL: TS1 и TS2 (рис. 8)

TS1 –для управления светофором A

TS2 –для управления светофором B

По умолчанию схема состоит из одного шага *INIT*. Добавим еще 4, как показано на рис. 9.

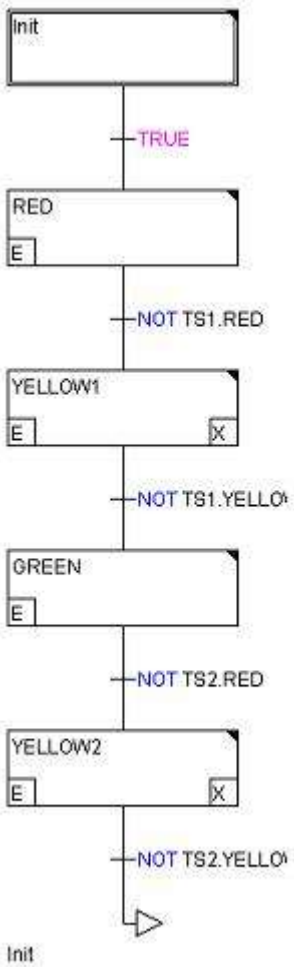

Рис.9. PLC_PRG

© Компоненты и технологии N7 2005г

Каждый шаг описывает одной стадию работы светофоров.

Начнем с шага *INIT*:

```
TS1.GREEN_DELAY:=T#6s;
TS1.RED_DELAY:= T#6s;
TS2.GREEN_DELAY:= T#6s;
TS2.RED_DELAY:= T#6s;
```

Здесь мы задали время включения каждого из сигналов светофоров.

Следующий шаг *RED* включает красный сигнал светофора A и зеленый сигнал светофора B. Для реализации этого шага нам понадобиться два действия – входное и основное. Входное действие выполняется один раз при входе в шаг, основное - в каждом цикле, пока не будет достигнуто условие перехода. Во входном действии мы просто включаем нужные сигналы светофора:

```
TS1(SWITCH_GREEN:=FALSE,SWITCH_YELLOW
:=FALSE,SWITCH_RED:=TRUE);
TS2(SWITCH_GREEN:=TRUE,SWITCH_YELLOW:
=FALSE,SWITCH_RED:=FALSE);
```

В основном действии функциональный блок *TS1* вызывается периодически, до тех пор пака таймер не выключит красный сигнал светофора A.

Шаг *Yellow1* состоит из 3 действий. Входной шаг включает желтый сигнал светофора A:

```
TS1(SWITCH_GREEN:=FALSE,SWITCH_YELLOW
:=TRUE,SWITCH_RED:=FALSE);
```

В основном шаге вызывается TS1 до тех пор пока желтый сигнал A не выключиться.

Выходное действие выполняется, когда переход уже ‘сработал’. В нем выключается зеленый сигнал светофора B:

```
TS2(SWITCH_GREEN:=FALSE);
```

Шаг *GREEN* включает зеленый сигнал светофора A, и красный сигнал светофора B. Для этого во входном действии шага прописываем:

```
TS1(SWITCH_GREEN:=TRUE,SWITCH_YELLOW:
=FALSE,SWITCH_RED:=FALSE);
TS2(SWITCH_GREEN:=FALSE,SWITCH_YELLOW
:=FALSE,SWITCH_RED:=TRUE);
```

В основном действии:

```
TS2;
```

Последний шаг *YELLOW2* включает желтый сигнал светофора B и выключает зеленый сигнал светофора A.

Входное действие *YELLOW2*:

```
TS2(SWITCH_GREEN:=FALSE,SWITCH_YELLOW
:=TRUE,SWITCH_RED:=FALSE);
```

Основное действие *YELLOW2*:

```
TS2;
```

Выходное действие YELLOW2

```
TS1(SWITCH_GREEN:=FALSE);
```

Описанная последовательность шагов выполняется циклически.

Теперь добавим в нашу программу возможность включения/выключения светофора. Желтый сигнал выключенного светофора должен мигать. Выключение светофора можно проводить только после завершения полного цикла работы светофора.

За включение/выключение светофора будет отвечать глобальная логическая переменная ON.

Добавим в нашу SFC-схему альтернативную ветвь, которая выполняется, когда светофор выключен (ON = FALSE).

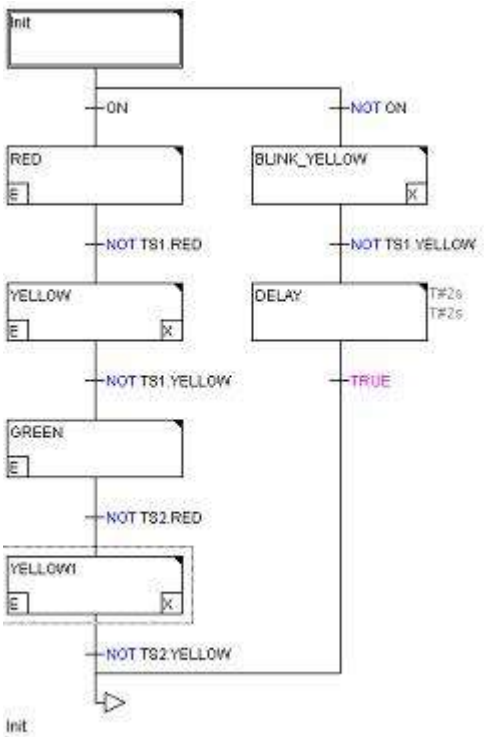

Рис. 10. PLC_PRG с альтернативными ветвями

В альтернативной ветви есть два шага. Шаг *BLINK_YELLOW* включает желтый сигнал обоих светофоров на две секунды:

```
TS1(SWITCH_YELLOW:=TRUE);
TS2(SWITCH_YELLOW:=TRUE);
```

После того как желтые сигналы будут выключены, в выходном действии сбрасываются входные сигналы:

```
TS1(SWITCH_YELLOW:=FALSE);
TS2(SWITCH_YELLOW:=FALSE);
```

Это сделано для того, чтобы в следующем цикле обеспечить передний фронт на этих входах.

Шаг *DELAY* пустой. Он нужен только для задержки, которая длиться в течении 2с. В течение этого времени желтый сигнал светофора не горит. Для задания этой задержки нужно в свойствах шага определить параметр *Minimum time* (рис. 11)

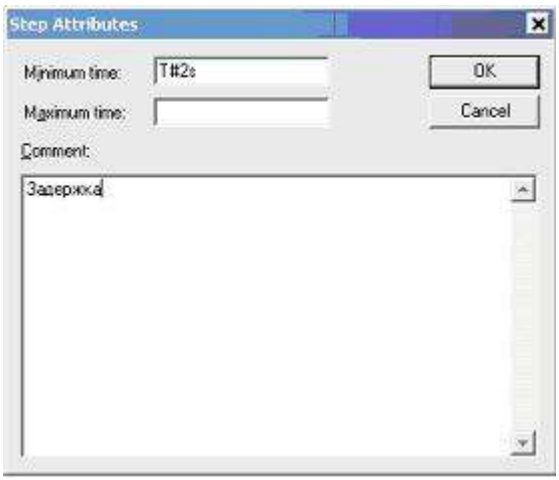

Рис. 11 Настройка времени шага

Осталось лишь модифицировать программу так, чтобы можно было задавать период работы светофора и время включения зеленого сигнала светофора. Будем считать, что период работы светофора можно задавать от 20 секунд до 6 минут. Достаточно лишь задавать время включения зеленого сигнала светофора А, так как время включения красного можно посчитать через период.

Время зеленого сигнала В будет соответствовать времени красного сигнала А, а время красного В – зеленому А. Это связано с тем, что светофоры работают в противофазе.

Период работы светофора будет храниться в переменной *TS_period*, а время разрешающего сигнала светофора А (в процентах от периода) в переменной *TS1_greentime*.

Установку времен включения сигналов обоих светофоров будем проводить в шаге *INIT*.

```
TS1.GREEN_DELAY:=(TS_period*TS1_greentime)/
100;
TS1.RED_DELAY:=TS_period-T#4s-
TS1.GREEN_DELAY;
TS2.GREEN_DELAY:=TS1.RED_DELAY;
TS2.RED_DELAY:=TS1.GREEN_DELAY;
```

Теперь, когда логика работы нашей системы полностью описана, необходимо связать наше приложение с внешним миром – связать имена переменных с входами/выходами контроллера и создать интерфейс, с помощью которого диспетчер будет управлять светофором.

Настройка входов/выходов

Начнем с определения входов/выходов. Обычно входы/выходы объявляются с помощью ПЛК конфигуратора (рис. 12).

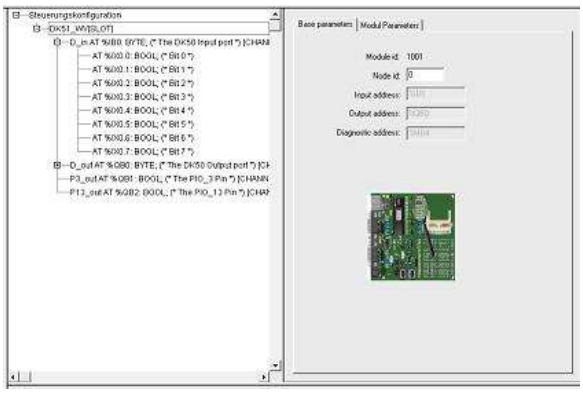

Рис. 12. ПЛК конфигуратор

По МЭК-адресу объявляется глобальная переменная, которая в проекте будет являться входом или выходом. Но мы пойдем другим путем. Дело в том, что переменные, которые у нас соответствуют физическим сигналам светофора объявлены внутри функциональных блоков. То есть, чтобы вывести их значения на выходы контроллера, значения этих переменных нужно скопировать в объявленные в ПЛК конфигураторе глобальные переменные. Это не очень удобно. Поэтому мы воспользуемся конфигурационными переменными.

Для этого в функциональном блоке *TRAFFICSIGNAL* нужно модифицировать объявления выходов функционального блока следующим образом:

```
VAR_OUTPUT
    GREEN       AT %Q*:BOOL;
    YELLOW      AT %Q*:BOOL;
    RED         AT %Q*:BOOL;
END_VAR
```

Эта запись говорит о том, что эти выходы функционального блока будут являться выходами контроллера. Теперь определим какими именно.

Для этого в глобальных переменных проекта в списке *variable_configuration* напишем следующее:

```
PLC_PRG.TS1.RED     AT %QX0.0:BOOL;
```

© Компоненты и технологии N7 2005г

PLC_PRG.TS1.YELLOW AT %QX0.1:BOOL;
PLC_PRG.TS1.GREEN AT %QX0.2:BOOL;

PLC_PRG.TS2.RED AT %QX0.3:BOOL;
PLC_PRG.TS2.YELLOW AT %QX0.4:BOOL;
PLC_PRG.TS2.GREEN AT %QX0.5:BOOL;

Эта запись говорит о том, что *PLC_PRG.TS1.RED* соответствует нулевому выходу контроллера, *PLC_PRG.TS1.YELLOW* – первому и т.д. То есть когда значение переменной *PLC_PRG.TS1.RED* равно *TRUE* нулевой выход будет активен. Аналогично и для других выходов.

Создание визуализации

Осталось только создать интерфейс пользователя – и наша система управления движением на перекрестке будет готова. Для этого нам нужно создать визуализацию на вкладке CoDeSys *Visualization*. Создадим визуализацию с именем *PLC_VISU*.

Теперь нужно нарисовать форму. Начнем со светофора. Светофор – это три окружности и один прямоугольник – все очень просто (рис. 13)

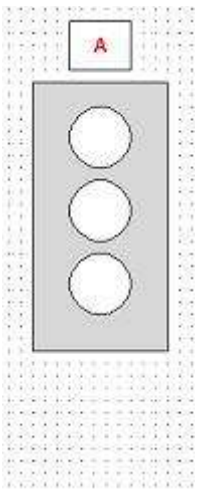

Рис. 13. Визуализация светофора

Теперь нужно настроить созданные объекты. Тут идея такая - каждый объект связывается с логической переменной проекта. Если переменная имеет значение FALSE , то объект имеет один цвет, TRUE – другой (этот цвет называется *alarm color*). Продемонстрируем, как настроить один из сигналов светофора:

1. Дважды кликаем по объекту
2. В меню выбираем пункт *Variables* и вписываем в поле *Change color* имя переменной (рис. 14). Можно просто нажать F2 и выбрать переменную с помощью Ассистента ввода.

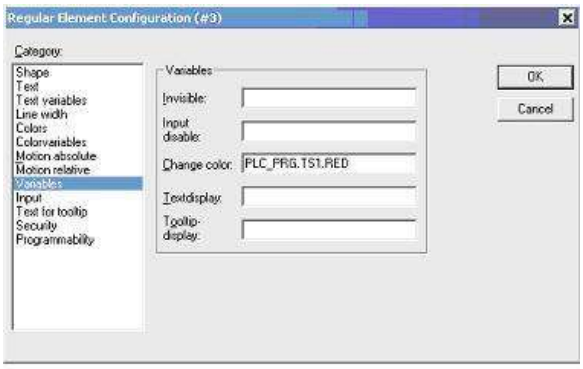

Рис. 14. Настройка свойств сигнала светофора

3. Настроим цвета в пункте Colors

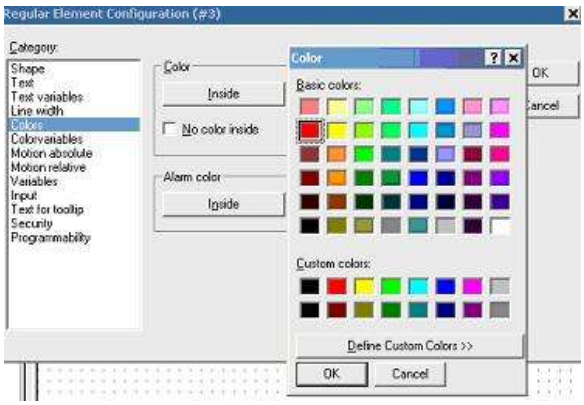

Рис. 15. Настройка свойств сигнала светофора

Alarm color будет красным, а *color* – белым.

Аналогично настраиваются другие сигналы светофора.

Теперь клонируем (выделяем и копируем) созданный светофор, изменяем имена переменных (меняем TS1 на TS2) и наши светофоры готовы.

Приступим к созданию блока управления светофором. Во-первых, нам нужна кнопка включения/выключения светофора. Это обычный прямоугольник. Настраивается он примерно так же, как и сигналы светофора. Различие в том, что он предназначен для ввода информации, поэтому нужно указать какая переменная будет изменяться при нажатии на эту кнопку. Как это сделать показано на рис. 16.

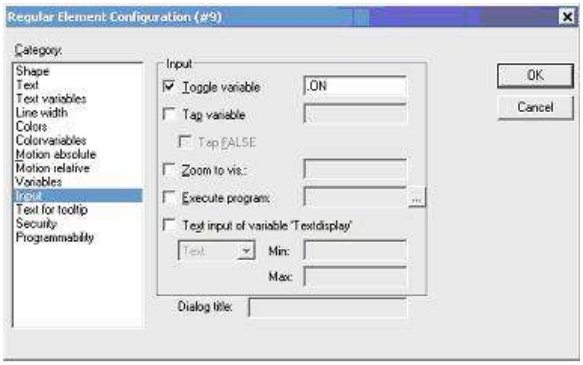

Рис. 16. Контекстное меню для настройки кнопки включения светофора

© Компоненты и технологии N7 2005г

У нас есть светофоры, мы можем их включать и выключать – осталось только создать поля для ввода периода работы светофора и времени включения разрешающего сигнала светофора А.

Делается это следующим образом:

1. Создаем прямоугольник.
2. В его свойствах в пункте *text* делаем следующие настройки (См. рис.17).

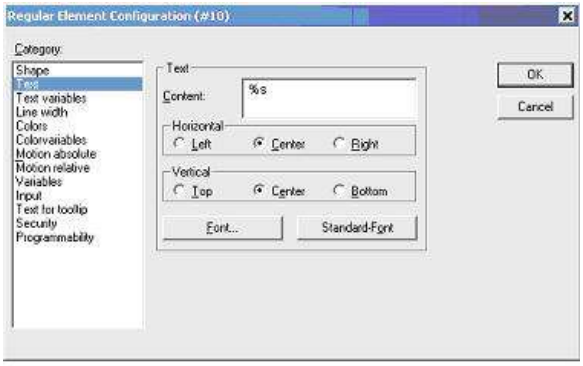

Рис. 17. Настройка поля ввода

3. В пункте *Variables* указываем имя переменной (См. рис. 18)

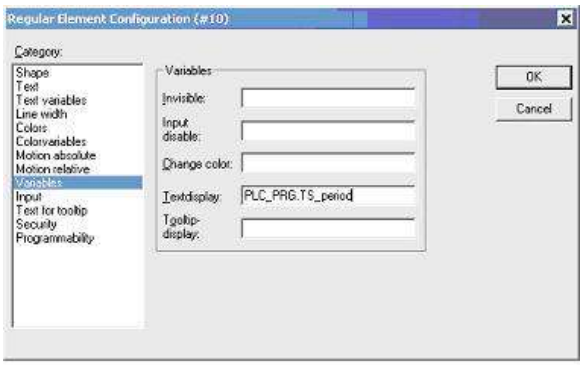

Рис. 18. Настройка поля ввода

4. Теперь внутри созданного прямоугольника будет отображаться переменная *PLC_PRG.TS_period*. Настраиваем пункт *Input* (рис. 19)

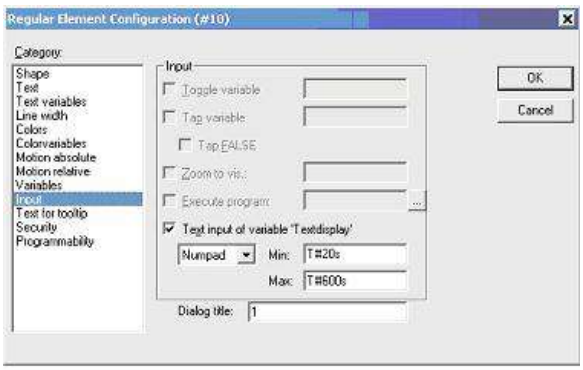

Рис. 19. Настройка поля ввода

Сделанные настройки указывают на то, что период работы светофора будет вводиться с экранной клавиатуры, и будет лежать в диапазоне от 20 до 600с

Поле ввода для времени включения разрешающего светофора А настраивается аналогично, только в качестве переменной *textdisplay* используется переменная *TS1_greentime*, а величина задается в диапазоне от 10 до 90.

В итоге должна получиться картинка, показанная на рисунке 20.

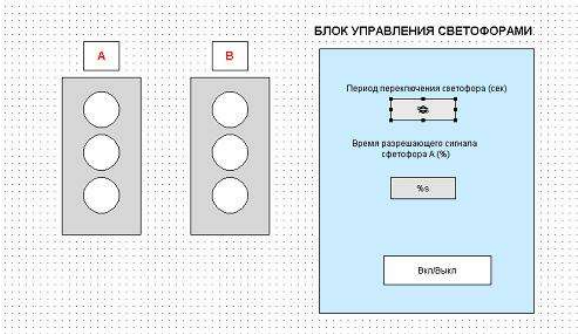

Рис. 20. Визуализация

Как только мы соединимся с контроллером, загрузим и запустим проект, светофоры оживут, и на дорогах воцарится порядок.

Как сделать этот последний шаг?

Настройка связи с контроллером

Открываем меню *Online/Communication parameters* и создаем TCP/IP соединение, в котором нужно указать **ip-адрес** контроллера (рис. 21)

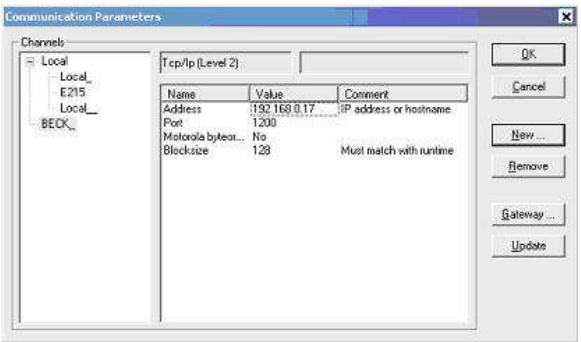

Рис. 21. Настройка параметров связи с BECK IPC@CHIP

И наконец, выполняем команду **Login**, затем **Run** и дело сделано.

© Компоненты и технологии N7 2005г

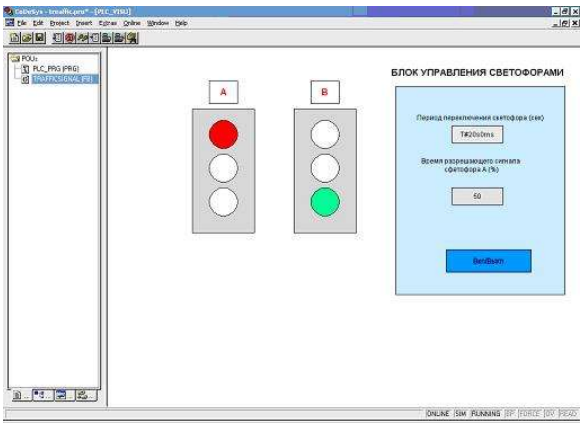

Рис. 22. Визуализация в режиме Online

Мы можем управлять светофором с помощью визуализации через Интернет, как бы далеко он от нас не находился.

Согласитесь, что создать визуализацию для управления контроллером в CoDeSys гораздо проще, нежели написать специальное cgi – приложение.

Мы коснулись только самых простых возможностей использования CoDeSys в собственных устройствах. Описания промышленных систем, использующих CoDeSys, и практические примеры их применения вы найдете страницах сайта Automation Alliance: http://www.automation-alliance.com.

Надеемся, что наше краткое введение поможет вам в практической реализации собственных интересных идей. Мы – постоянные читатели журнала будем рады, если вы найдете возможность рассказать нам о полученных результатах.

Литература

1. Петров И.В. Пастушенков Д.В, IPC@CHIP или маленькие ядра для большой артиллерии 2, Компоненты и технологии. N6, 2005.

© Компоненты и технологии N7 2005г